# COMSC-210 Lab 14
# O(n log n) Sorting Algorithms

**GOOD PROGRAMMING PRACTICES** *show / hide*

**ABOUT THIS ASSIGNMENT**
In this assignment you will implement O(n log n) in your DynamicArray class template.

After you complete this lab assignment, post the required files to the COMSC server so that you receive the proper credit for this 50-point lab. Your posted files are due at midnight of the evening of the due date indicated in the course outline. Use the "Submit your FA2015 work" link on the class website to post your file for this lab, using the **lab14** folder provided.

**LAB 14a: Write A Sortable Array Class Template** [ `SortableDynamicArray.h` and `MySortableDynamicArray.cpp` ]
**Purpose**. The purpose of this lab is twofold: (1) for you to implement O(n log n) sorting of an array, and (2) for you to implement a binary search algorithm. The resulting template can be used in any program in place of a C++ array whose data type supports both operator-less-than and equals-equals.

**Requirements.** Rewrite `DynamicArray.h` and `MyDynamicArray.cpp` from lab 4a as **`SortableDynamicArray.h`** and **`MySortableDynamicArray.cpp`**, adding a sort function. Name the new templated class `SortableArray`. Do NOT use inheritance -- do copy/paste/mark-up.

Use this prototype for a **public** sort function:

```
void sort(int);
```

...implementing either heapsort, quicksort, or mergesort from the lecture, *your choice*. The `int` parameter specifies how many elements to sort, starting from element zero. That is, `sort(10)` should sort only elements [0] through [9].

Add a public binary search function with this prototype:

```
int bsearch(int, const T&) const
```

...to perform a "binary search" as explained in lecture. The `int` parameter specifies how many elements to search, starting with element zero. It should return the index of the found value as it appears in the sorted array. Note that it's the responsibility of the user of your template to use binary search only after the array's been sorted, so you don't have to sort or validate in your function.

Note that sorting and searching will not work well if there are "holes". So your sort function should first check if there are holes in the range to be sorted, and do nothing if there are any holes. Same for your search function -- if there are holes in the range, return a negative one for the index.

Improve the app so that after all data entry is complete, it:

1. outputs the size of the SortableDynamicArray object,
2. outputs the number of values actually entered by the user (which should match the above),

3. outputs the unsorted list,
4. implements key lookups in a user-controlled loop, allowing multiple look-ups until the user elects to stop, and
5. **prompts the user how many values to include in sorting and the binary searching,**
6. sorts the data entries made by the user, lo-to-hi **up to the number enterd above**, and outputs them, and
7. **implements binary search lookups in a user-controlled loop, allowing multiple lookups until the user elects to stop**.

Submit the two files (1 CPP and 1 H) to the class website for credit.

**Program I/O.** Input: Same as lab 6c, plus the binary search loop. Output: Same as lab 6c, plus sorting and binary search results.

---

**LAB 14b: Applying A Data Structure To A Database Program** [ `DvcSchedule14b.cpp` ]
**Purpose**. In this lab you will apply a O(n log n) sort function to a database program.

**Requirements.** Rewrite `DvcSchedule13b.cpp` from lab 5b as **`DvcSchedule14b.cpp`**, but instead of using the SortedStaticArray, use your new SortedDynamicArray and its sort function.

The next and final version of this program will use the STL `map`, in which case there is no need to call a sort function. But do NOT use the STL `map` for this version.

Submit the CPP to the class website for credit. Do NOT submit the TXT input file.

**Program I/O.** Same as lab 5b.

---

**How to pause a console program:** *show / hide*

---

**GRADING RUBRIC** *show/hide*

---